# Post-Silicon Fault Localisation Using Maximum Satisfiability and Backbones

Charlie Shucheng Zhu Georg Weissenbacher Sharad Malik
Department of Electrical Engineering, Princeton University

***Abstract*—The localisation of faults in integrated circuits is a challenging problem and a dominating factor in the overall verification effort. Electrical bugs, in particular, surface only in the fabricated prototypes, leading to behaviour deviating from the *golden model*. Limited observability complicates their localisation: Logging mechanisms such as trace buffers allow us to retain only a limited execution history.**

**A symbolic analysis of the RTL design can find discrepancies between the values recorded in the trace buffer and the intended behaviour. Contemporary MAX-SAT solvers are then able to identify a maximal subset of the RTL design that is consistent with the observed behaviour. The elements in the *complement* of this subset represent potential locations of the fault.**

**The scalability of contemporary decision procedures dictates the size of a *window* of execution cycles which we can analyse using symbolic techniques. Current MAX-SAT-based fault localisation techniques require this window to span the fault as well as the error it causes. To address the scalability issues resulting from large window sizes, we propose to *slide* a smaller window along the temporal axis, constraining it with the information recorded in the trace buffer for the respective execution cycles.**

**In this scenario, the localisation attempt may fail: The limited information provided by the trace buffer may be insufficient to pin down the exact temporal and spatial location of the fault. We propose to use *backbones* to identify information that can be propagated across *sliding windows*. The backbone of a symbolic representation of a circuit is the set of signals that are immutable under the given constraints (e.g., the output and trace buffer values). This additional information has several benefits: Firstly, it may be instrumental in locating the fault. Secondly, it may enable a reduction of the size of the of trace buffers and the sliding window. Our preliminary experimental results demonstrate that the use of backbones allows us to reduce the size of the sliding windows or the trace buffer.**

## I. Introduction

The localisation of faults in fabricated prototypes, referred to as *silicon debug* or *post-silicon validation*, is a challenging and time-consuming problem. According to [1], the temporal and spatial isolation of a fault "typically dominate[s] the effort expended during the debug process for a bug." One of the aspects that distinguishes post-silicon validation from simulation or formal verification of the RTL design is the ability to execute long test scenarios. This comes at the cost of *limited observability* of signals in integrated circuits. Logging techniques such as trace buffers enable us to track a relatively small number of signals over a limited amount of time (e.g., a few thousand execution cycles). If a test case reveals an error, this limited execution history has to suffice to locate the fault causing the erroneous behaviour of the chip.

To locate the fault, we require sufficient information about the execution history to reconstruct the scenario that led to the error. Such information can be obtained by augmenting the processor with hardware recorders (referred to as *trace buffers*) that keep track of limited information to enable the reconstruction of the instruction sequence leading up to the error. An architecture-specific analysis such as IFRA [2] can then be used to narrow down the location of the fault. The advantage of this approach is that the reproduction of the failure is not required for localisation purposes. The design of such an analysis, however, requires considerable insight and needs to be adapted for individual processor architectures.

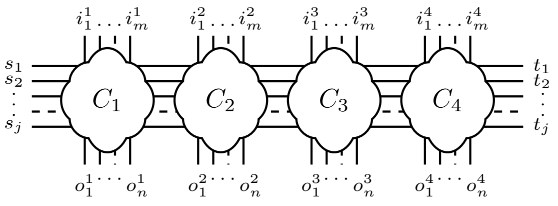


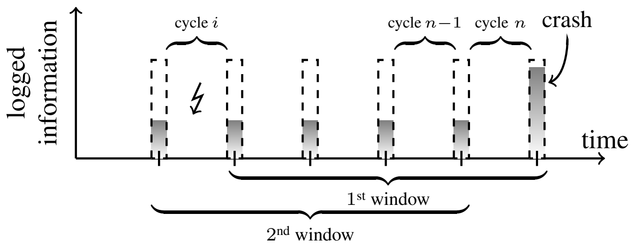
Fig. 1: Unfolded circuit encoding four execution cycles

We propose an adaptation of a fault-localisation technique which is *architecture independent* and has been successfully applied for fault diagnosis [3] as well as design debugging [4], [5], [6], [7]. Given the RTL design in a language such as Verilog, it is possible to construct a symbolic representation of $k$ execution steps by unfolding the combinational logic $C$ of the sequential circuit. The unfolding yields an *iterative logic array* [8] as illustrated in Figure 1. The resulting formula encodes *all correct* executions within a *window* of length $k$.

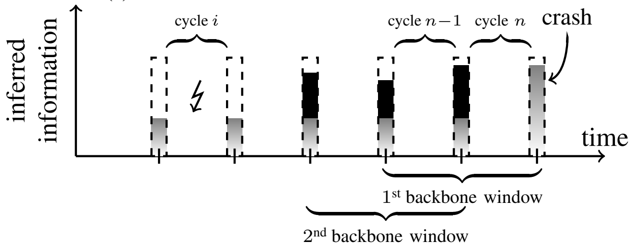
To identify the spatial and temporal location at which the behaviour of the device under test deviates from that of the *golden model*, we constrain the symbolic representation with the values recorded in the trace buffers. If the resulting formula is unsatisfiable, the fault must have occurred within the given window. Using the techniques presented in [9], [10], one can then compute the maximal subsets of the circuit which are *consistent* with the observed behaviour of the integrated circuit. The *complements* of these subsets (known as *minimal correction sets*) identify potential locations of the fault.

The success of this technique hinges on the size of the window and the information recorded in the trace buffers. The former is dictated by the scalability of the underlying decision procedure. The latter is determined by practical issues such as cost and required performance of the integrated circuit.

Figure 2a illustrates the situation in which a transient electrical fault in execution cycle $i$ causes an error after cycle



(a) Error localisation with limited window size



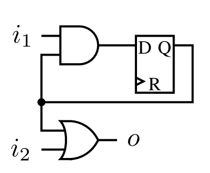
(b) Additional state bits inferred using backbones

Fig. 2: Error localisation with and without backbones

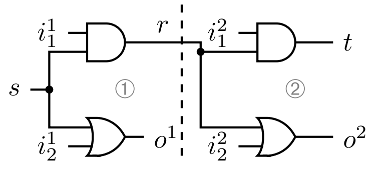
$n$. The dashed bars represent the state bits of the circuit for each execution step, and the shaded area indicates the fraction of those bits recorded in the trace buffer. The diagram suggests that we have near-complete information about the state bits in the *crash state* (obtained by means of scan chains [11]), but very limited information about what happened before the error occurred. The curly brackets at the bottom of Figure 2a illustrate two attempts to use a window of size four to locate the fault. The first attempt fails because the fault does not lie within the window. The second attempt fails because the information recorded in the trace buffer is insufficient to derive a contradiction: By shifting the window to the left, the analysis drops crucial information about the crash state.

Figure 2b illustrates the use of *backbones* to infer additional information that can be propagated across the sliding windows. The backbone of an unfolding of a circuit is the set of signals that are immutable under the respective constraints (i.e., the bits of the crash state and the trace buffer values). For instance, the backbone of the formula $(x \oplus y) \cdot (x + z)$ under the constraint $x \mapsto \mathbf{1}$ is $\{x \mapsto \mathbf{1}, y \mapsto \mathbf{0}\}$. The backbone of an unfolding (possibly smaller than the window used for localisation) that is constrained by the recorded state bits potentially provides additional state bits (indicated by the black bars in Figure 2b) which can be subsequently used in overlapping windows. Thus, backbones (which can be computed using the algorithms in [12]) can be used to propagate information from the crash state backwards in time to earlier cycles. This additional information can be crucial to fault localisation. We emphasise that our analysis is static and therefore not restricted to reproducible permanent faults.

**Contributions:** We address the scalability limits of symbolic reasoning by *sliding* an analysis window of fixed size along the temporal axis. We present a novel application of *backbones* allowing us to reduce the loss of information resulting from *limited observability* in post-silicon validation.



(a) A sequential circuit



(b) A 2-cycle unfolding

| | | |
|---|---|---|
| cycle ① | $(\overline{r}\, i_1^1)\, (\overline{r}\, s)\, (\overline{i_1^1}\, \overline{s}\, r)$ | $(\overline{i_2^1}\, o^1)\, (\overline{s}\, o^1)\, (\overline{o^1}\, i_2^1\, s)$ |
| cycle ② | $(\overline{t}\, i_1^2)\, (\overline{t}\, r)\, (\overline{i_1^2}\, \overline{r}\, t)$ | $(\overline{i_2^2}\, o^2)\, (\overline{r}\, o^2)\, (\overline{o^2}\, i_2^2\, r)$ |

(c) CNF encoding of unfolded circuit. As usual, the operators $+$ and $\cdot$ are dropped for compactness. The clauses are grouped with respect to the gates and cycles by which they are contributed.

Fig. 3: A simple example

## II. Fault Localisation Using Max-SAT

In the following we use propositional operators $(\cdot, +, \oplus)$ and atoms $(i, o, s, \ldots)$ to represent gates and signals, respectively. As usual, a literal is either an atom or its negation, a clause is a sum of literals, and a formula in conjunctive normal form (CNF) is a product of clauses. Every unfolded combinational circuit has an equi-satisfiable representation in CNF which can be obtained in polynomial time [13]. Each instance of a gate in the unfolded circuit corresponds to a group of clauses in the corresponding CNF representation. We use $C_i$ to denote the CNF instantiation of the combinational logic of the RTL design representing the $i^{\text{th}}$ execution cycle (c.f. Figure 1). Similarly, we use $T_i$, a conjunction of literals, to represent the state bits recorded by the trace buffer after the $i^{\text{th}}$ execution step. Note that $T_i$ is simply $\mathbf{1}$ if no state bits were recorded at that point. Moreover, we assume that $T_0$ and $T_n$ represent the information available about the initial and the crash state, respectively. Finally, $I_i$ and $O_i$ represent the input and output constraints of cycle $i$. Accordingly, the CNF formula

$$W_m^k \overset{\text{def}}{=} \bigwedge_{i=m}^{m+k-1} \underbrace{T_{(i-1)} \cdot I_i}_{\text{input constraints}} \cdot \underbrace{C_i}_{\text{circuit}} \cdot \underbrace{O_i \cdot T_i}_{\text{output constraints}} \qquad (1)$$

represents a *window* of $k$ consecutive execution cycles starting at cycle $m$ constrained with the corresponding inputs, outputs, and state bits recorded by the trace buffer.

Given an *unsatisfiable* instance of (1), the goal is to find the clauses (gates, respectively) that are most likely responsible for the fault. This can be achieved by identifying a *minimal correction set* (MCS), i.e., a *minimal* set of gates that need to be dropped from (1) such that the formula becomes satisfiable. This corresponds to finding the maximum set of clauses that are consistent, an NP-hard optimisation problem commonly known as Max-SAT. More specifically, we are only interested in dropping clauses that correspond to gates; the constraints introduced by trace buffer values and inputs or outputs are *hard constraints*. This specialisation of Max-SAT is known as *partial* Max-SAT. Algorithms to solve partial Max-SAT instances are discussed in [9] and [10], for instance.

Consider the sequential circuit in Figure 3a. After a reset of the flip-flop, we expect the output $o$ to remain $\mathbf{0}$ as long as the input signal $i_2$ is constantly $\mathbf{0}$. Assume, however, that we observe an output value of $\mathbf{1}$ after two cycles when executing the described scenario on the chip. Figure 3b depicts a two-cycle unfolding of the circuit. Figure 3c shows the corresponding CNF encoding. Assume that we observe and record the values $o^1 \mapsto \mathbf{0}$ and $o^2 \mapsto \mathbf{1}$ during a test-run with the initial state $s \mapsto \mathbf{0}$ and the stimuli $i_2^1 \mapsto \mathbf{0}$, and $i_2^2 \mapsto \mathbf{0}$. Note that we have no information about the signal $r$. These observations contribute the hard constraint $\overline{o^1} \cdot o^2 \cdot \overline{s} \cdot \overline{i_2^1} \cdot \overline{i_2^2}$, which is not satisfiable in conjunction with the formula in 3c. Using a MAX-SAT solver, we can derive that the conjunction becomes satisfiable if we drop either $(\overline{r}\, s)$ or $(\overline{o^2}\, i_2^2\, r)$ (both of which are an MCS) from 3c. Accordingly, either the AND-gate in cycle one or the OR-gate in cycle two must have defaulted.

This approach also addresses multiple faults by using fault cardinality constraints [3]. Existing MCS-based fault localisation techniques require the window $W_m^k$ (Formula 1) to span the fault as well as the crash state, which may result in a large formula exceeding the capabilities of the decision procedure.

In contrast, we *restrict the analysis to a window of fixed size* which we *slide* along the temporal axis. In the following section, we discuss a novel application of *backbones* allowing us to reduce the loss of information resulting from the limited window size.

## III. Propagating Information Using Backbones

We will now consider the case in which the scalability of the underlying decision procedure dictates a window size smaller than the number of cycles of the entire test run. For the purpose of an example, we revisit the scenario in Figure 3 and assume that the window size is limited to a single cycle. Observe that neither the first line of Figure 3c in conjunction with $\overline{o^1} \cdot \overline{s} \cdot \overline{i_2^1}$ nor the second line in conjunction with $o^2 \cdot \overline{i_2^2}$ yields a contradiction; the technique introduced in §II fails. The reason is that we lack crucial information, namely the value of the signal $r$. We propose the use of *backbones* to aid the reconstruction of this information.

Formally, the *backbone* of a satisfiable propositional formula $F$ comprises the values of all atoms $p$ in $F$ for which either $(\overline{F} + p)$ or $(\overline{F} + \overline{p})$ holds, i.e., $p$ takes the *same* value in *all* satisfying assignments of $F$. We use a SAT solver to compute an initial satisfying assignment and subsequently try to flip the value of each literal, thus changing the assignment. Literals whose values differ in subsequent assignments are not part of the backbone. This algorithm performs one call to the SAT solver per literal *in the worst case*. Only the first call has to solve the instance from scratch; the subsequent calls are incremental, making the algorithm practical on large scale circuits [12]. The backbone of the formula from our example

$$\underbrace{(\overline{t}\, i_1^2)\,(\overline{t}\, r)\,(\overline{i_1^2}\,\overline{r}\, t)\,(\overline{i_2^2}\, o^2)\,(\overline{r}\, o^2)\,(\overline{o^2}\, i_2^2\, r)}_{\text{cycle ②}}\ \underbrace{(o^2)\,(\overline{i_2^2})}_{\text{hard constraints}}\ ,$$

is $o^2 \mapsto \mathbf{1}$ $i_2^2 \mapsto \mathbf{0}$, and $r \mapsto \mathbf{1}$. This backbone provides a value for the previously unknown signal $r$. In general, backbones *under-approximate* the information available to the SAT solver in the analysed time-frame.

Effectively, this computation propagates the error encoded in the constraint $(o^2)\,(\overline{i_2^2})$ backwards. By propagating the newly learnt information to cycle ① we obtain the desired contradiction:

$$\underbrace{(\overline{r}\, i_1^1)\,(\overline{r}\, s)\,(\overline{i_1^1}\,\overline{s}\, r)\,(\overline{i_2^1}\, o^1)\,(\overline{s}\, o^1)\,(\overline{o^1}\, i_2^1\, s)}_{\text{cycle ①}}\ \underbrace{(\overline{o^1})\,(\overline{s})\,(\overline{i_2^1})}_{\text{constraint}}\ \underbrace{(r)}_{\text{backbone}}$$

As expected, a MAX-SAT solver is able to determine that $(\overline{r}s)$ must be dropped, and that the AND-gate in cycle ① is a potential culprit. Notably, we missed the second fault candidate due to the limited window size and the resulting lack of information.[1] We emphasise, however, that in the absence of the information provided by the backbone, the approach described in §II is unable to diagnose the fault altogether.

In general, we use the information provided by the backbone of an instance of Formula (1) for a given $m$ and $k$ to augment the trace buffer (c.f. Figure 2b) of windows overlapping the interval of cycles $[m-1, m+k)$. To this end, we compute the *largest* conjunctions $B_{(m-1)}, \ldots, B_{(m+k-1)}$ of literals over the signals in the respective cycles which are implied by Formula (1). Accordingly, $B_i \Rightarrow T_i$ for $i \in [m-1, m+k)$. We use $B_m^k \overset{\text{def}}{=} \bigwedge_{i=m}^{m+k} B_{(i-1)}$ to denote the backbone of $W_m^k$. Let $W_m^k$ and $W_n^l$ be two *satisfiable* windows with overlap $o$ (i.e., $n = m + k - o$ and $0 \leq o \leq \min(k, l)$). Then, $W_m^k \cdot W_n^l = W_m^{(k+l-o)}$, according to Formula (1). Now assume that $W_m^{(k+l-o)}$ is unsatisfiable but too large for a MAX-SAT solver. Using backbones, we *approximate* $W_n^l$ using $B_n^l$, i.e., $W_n^l \Rightarrow B_n^l$. The information in $B_n^l$ can enable the localisation of faults that $W_m^k$ failed to reveal. We construct $\widehat{W}_m^k = W_m^k \cdot B_n^l$. If $\widehat{W}_m^k$ is unsatisfiable, we use the approach described in §II to find the gates that need to be dropped from $W_m^k$ to make $\widehat{W}_m^k$ satisfiable. Since $W_m^k \cdot W_n^l \Rightarrow \widehat{W}_m^k$, these gates are necessarily a subset of the gates that need to be dropped to make $W_m^k \cdot W_n^l$ (i.e., the larger window $W_m^{(k+l-o)}$) satisfiable.

## IV. Experimental Results

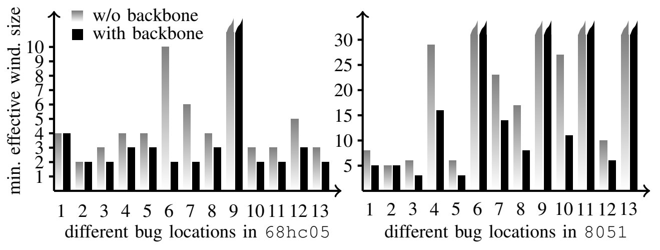
We evaluated our approach using the `68hc05` and `8051` processor designs used as case study in [14] (obtained from opencores.org). We converted the Verilog RTL designs into the DIMACS CNF format using the following translation steps:

$$\text{Verilog} \xrightarrow{\text{Altera Quartus}^2} \texttt{blif} \xrightarrow{\text{ABC}^3} \texttt{aig} \xrightarrow{\text{AIGER}^4} \texttt{cnf}$$

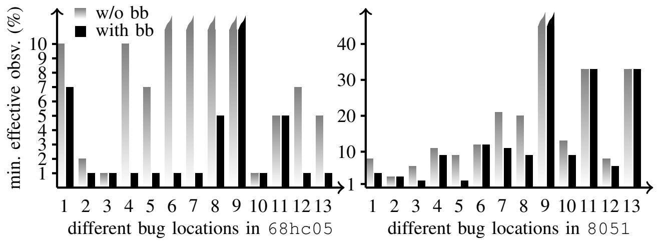
We randomly injected permanent *stuck-at-constant* faults (though we emphasise that our approach supports arbitrary fault models) into the RTL designs and obtained 26 failing test scenarios (13 for each design) of 2000 cycles length by means of SAT-based symbolic simulation. In each test scenario, we injected one fault at a time for the smaller `68hc05` design

[1]Note though, that a forward analysis yields the second fault candidate in our example: The backbone of cycle ① under the constraint $\overline{o^1} \cdot \overline{s} \cdot \overline{i_2^1}$ yields $r \mapsto \mathbf{0}$, which is inconsistent with the observations in cycle ②.

[2]altera.com [3]www.eecs.berkeley.edu/~alanmi/abc/ [4]fmv.jku.at/aiger/



(a) Reduction of window size with backbones (fixed buffer size 5%)



(b) Reduction of trace buffer size with backbones (using a fixed window size of 3 for the `68hc05` and 5 for the `8051` design)

Fig. 4: Experimental evaluation of backbones

and five faults at a time for the `8051` logic. All faults surface up to 1000 cycles before the end of the trace.

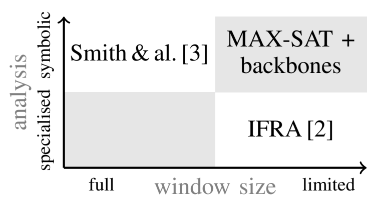
We recorded different percentages of the latches (chosen at random) in the trace buffer. We used the tool CAMUS [10], which implements the fault localisation algorithm discussed in §II, and the *iterative SAT-testing* algorithm described in [12] to compute backbones. The window size $k$ is the same for computing backbones and localisation. Our implementation uses a fixed overlap of $o = k - 1$. We slid the window backwards in time along the temporal axis of the test scenarios, propagating the backbones from previously analysed windows.

Using this setup, we ran two experiments: We *(a)* determined the minimum window size required to detect a fault for a fixed trace buffer size of 5% (Figure 4a) and *(b)* fixed the window size and increased the size of the trace buffer until the fault could be detected (Figure 4b). (Spiked bars indicate values that are off the scale.) We found that backbones enable a significant reduction of *(a)* the window size as well as *(b)* the size of the trace buffer required to locate faults.

## V. Related Work

*Instruction footprints* [2] (c.f. §I) enable the localisation of faults by providing sufficient information about the execution. This approach requires a design dependent localisation analysis which needs to be adapted for individual architectures.

Smith et al. [3] describes a technique similar to the one in §II. The approach covers multiple faults and different fault models, but requires the *window* to span all cycles of the test run. The work does not address limited observability.



The figure to the right relates our approach and the work presented in [2] and [3]. Our approach addresses limited observability and window-size using a generic SAT-based analysis. Yang et al. [15] proposes a SAT-based technique that, given a test scenario that results in a failure, identifies signals that are relevant to the analysis of the failure and should therefore be recorded in (configurable) trace buffers. This technique could potentially be helpful to increase the size of the backbones.

Paula et al. [14] proposes to compute *signatures* of states to narrow down the set of predecessor states of the crash state, effectively enabling backwards stepping. This allows to identify the error in an earlier cycle in a subsequent test run. The approach requires the repeated reproduction of the failure, which renders the approach infeasible for the localisation of transient electrical faults (which our approach makes possible).

There is a number of papers based on the the approach described in §II that address *pre-silicon* debugging (with full observability) by constraining a *faulty* RTL model with *correct* input/output pairs (given as a specification) [4], [5], [6], [7].

## VI. Conclusion

We presented a novel fault localisation technique which addresses the limited observability in post-silicon validation and demonstrated its applicability using small case studies.

## References

[1] D. Josephson, "The good, the bad, and the ugly of silicon debug," in *Design Automation Conference*. ACM, 2006, pp. 3–6.

[2] S.-B. Park and S. Mitra, "Post-silicon bug localization for processors using IFRA," *Communications of the ACM*, vol. 53, February 2010.

[3] A. Smith, A. G. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *Transactions on CAD of Integrated Circuits and Systems*, vol. 24, no. 10, pp. 1606–1621, 2005.

[4] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah, "Improved design debugging using maximum satisfiability," in *Formal Methods in Computer-Aided Design*. IEEE, 2007, pp. 13–19.

[5] A. Sülflow, G. Fey, R. Bloem, and R. Drechsler, "Using unsatisfiable cores to debug multiple design errors," in *Great Lakes Symposium on VLSI*. ACM, 2008, pp. 77–82.

[6] Y. Chen, S. Safarpour, A. Veneris, and J. Marques-Silva, "Spatial and temporal design debug using partial MaxSAT," in *Great Lakes Symposium on VLSI*. ACM, 2009, pp. 345–350.

[7] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris, "Automated design debugging with maximum satisfiability," *Transactions on CAD of Integrated Circuits and Systems*, vol. 29, pp. 1804–1817, 2010.

[8] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital systems testing and testable design*. Computer Science Press, 1990.

[9] Z. Fu and S. Malik, "On solving the partial MAX-SAT problem," in *Theory and Applications of Satisfiability Testing*, ser. LNCS, vol. 4121. Springer, 2006, pp. 252–265.

[10] M. H. Liffiton and K. A. Sakallah, "Algorithms for computing minimal unsatisfiable subsets of constraints," *Journal of Automated Reasoning*, vol. 40, no. 1, pp. 1–33, 2008.

[11] M. J. Y. Williams and J. B. Angell, "Enhancing testability of large-scale integrated circuits via test points and additional logic," *IEEE Transactions on Computers*, vol. C-22, pp. 46–60, 1973.

[12] J. Marques-Silva, M. Janota, and I. Lynce, "On computing backbones of propositional theories," in *European Conference on Artificial Intelligence (ECAI)*. IOS Press, 2010, pp. 15–20.

[13] G. Tseitin, "On the complexity of proofs in poropositional logics," in *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, vol. 2. Springer, 1983.

[14] F. M. De Paula, M. Gort, A. J. Hu, S. J. E. Wilton, and J. Yang, "BackSpace: formal analysis for post-silicon debug," in *Formal Methods in Computer-Aided Design*. IEEE, 2008, pp. 5:1–5:10.

[15] Y.-S. Yang, B. Keng, N. Nicolici, A. G. Veneris, and S. Safarpour, "Automated silicon debug data analysis techniques for a hardware data acquisition environment," in *International Symposium on Quality of Electronic Design*. IEEE, 2010.